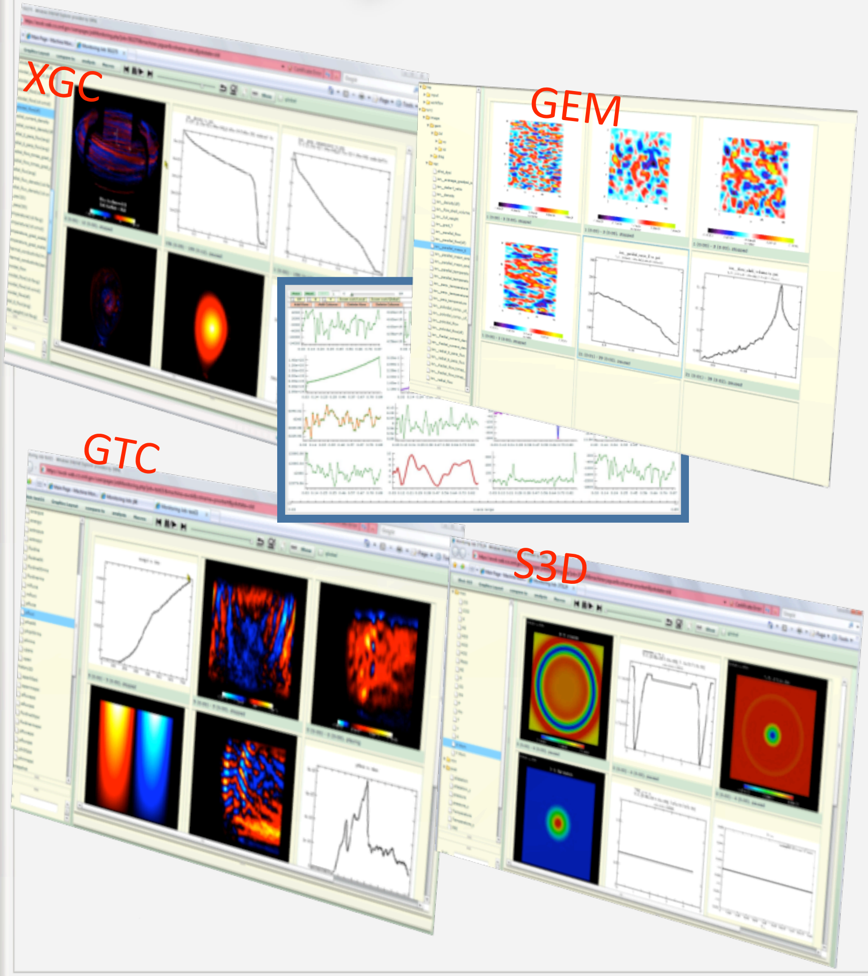
The Adaptable I/O System.

ADIOS



OLCF & NICS Spring Cray XT5 Hex-Core Workshop

Scott A. Klasky

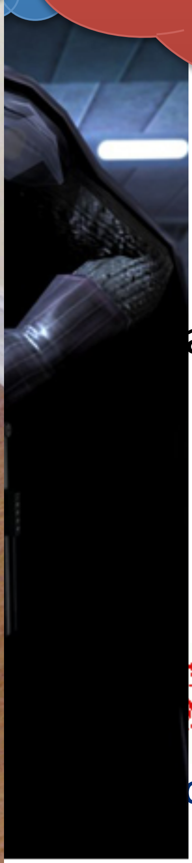
5/10/2010

Collaborators :

Jay Lofstead; Matthew Wolf; Karsten Schwan; Qing Liu; Norbert Podhorszki; Todd Kordenbrock; Ron Oldfield Hasan Abbasi; Fang Zheng; Ciprian Docan; Fan Zhang; Divya Dinakar; Roselyne Tchoua, Xiaosong Ma; Mladen Vouk, Nagiza Samatova; Alexander Romosan; Sriram Lakshminarasimhan; Michael Warren; Bing Xie; Arie Shoshani; Micah Beck; John Wu, Weikuan Yu, Yuan Tian, Stephane Ethier, Ray Grout, Seung Hoe Ku, Yong Xiao, Zhihong Lin

It's all about the applications.

But what about the I/O?



ankaran

ojects.

– Make it portable, scalable, FAST, reliable, accurate.

alen

ngs, G. Y.

M hours)

– C.S. Chang, S.H. Ku

GEM (30M hours)

– Y. Chen, S. Parker, W. Wang

Gysela5D (35M hours)

– P. Diamond, G. Dif Pradiler

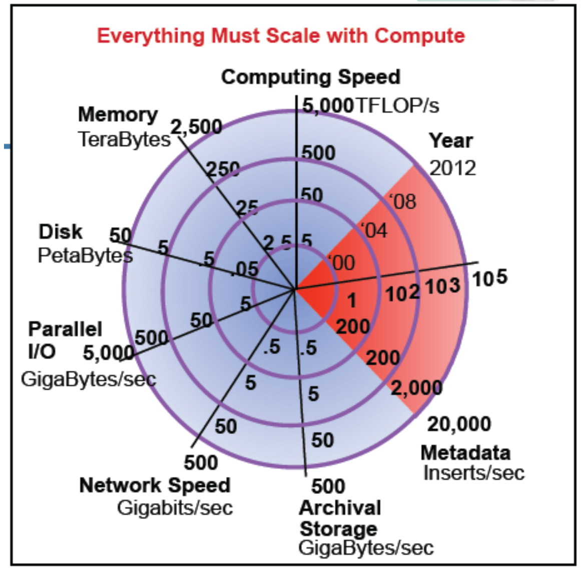
- **248 M hours**
- **16% INCITE time**

Advanced computing at ORNL-NCCS

Specs	Jaguar (XT4)	Jaguar (XT5)
Peak Pflops	0.3	2.3
Cores	31,328	224,256
Compute Nodes	7,832	18,772
Memory (TB)	60	300
Disk Bandwidth (GB/s)	72	120
Time to write memory to disk (s)	**853**	**2560**

File System, Problems for the Xscale

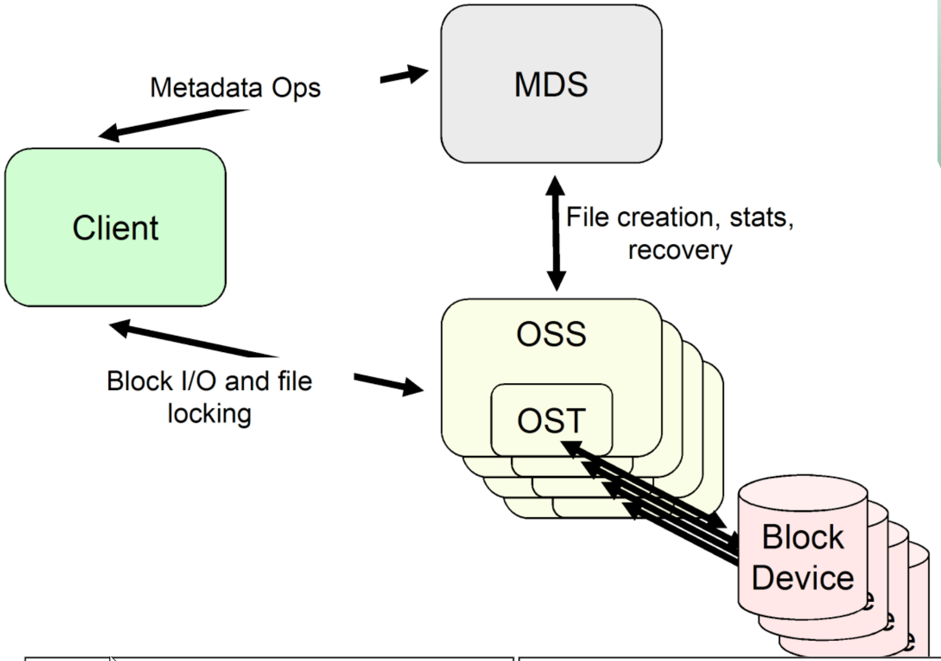
- The I/O on a HPC system is stressed
  - Checkpoint-restart writing
  - Analysis and visualization writing
  - Analysis and visualization reading
- Our systems are growing by 2x FLOPS/year.
- Disk Bandwidth is growing ~20%/year.
- Need the number of increase faster than the number of nodes
- As the systems grow, the MTF grows.
- As the complexity of physics increases, the analysis/viz. output grows.
- Need new and innovative approaches in the field to cope with this problem.

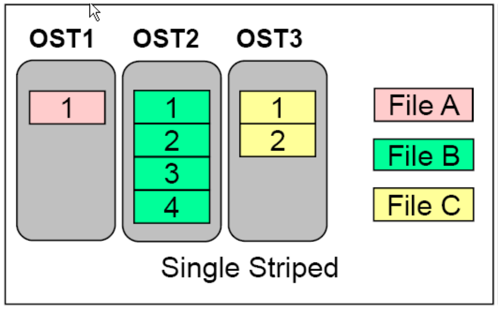


Garth Gibson 2010

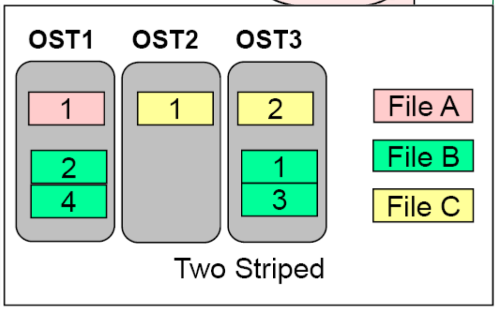
LUSTRE

- Lustre consists of four major components
  - MetaData Server (MDS)
  - Object Storage Servers (OSSs)
  - Object Storage Targets (OSTs)
  - Clients
- MDS
  - Manages the name space, directory and file operations
  - Stores file system metadata
  - Extended attributes point to objects on OSTs
- OSS
  - Manages the OSTs
- OST
  - Manages underlying block devices
- Striping, alignment, placement
  - Key for performance

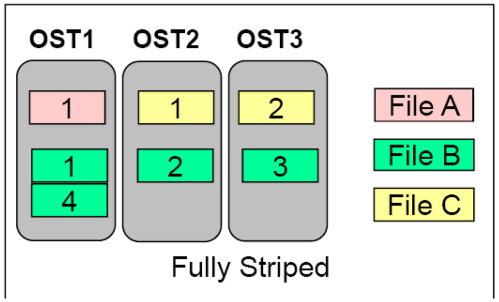




Single Striped



Two Striped



Fully Striped

- Stripe count (or width)
  - # of OSTs the file has been stripped over
- Stripe size
  - Size of each stripe on an OST
    - Normally same for all OSTs for a given file

I/O Efficiency and Simplicity

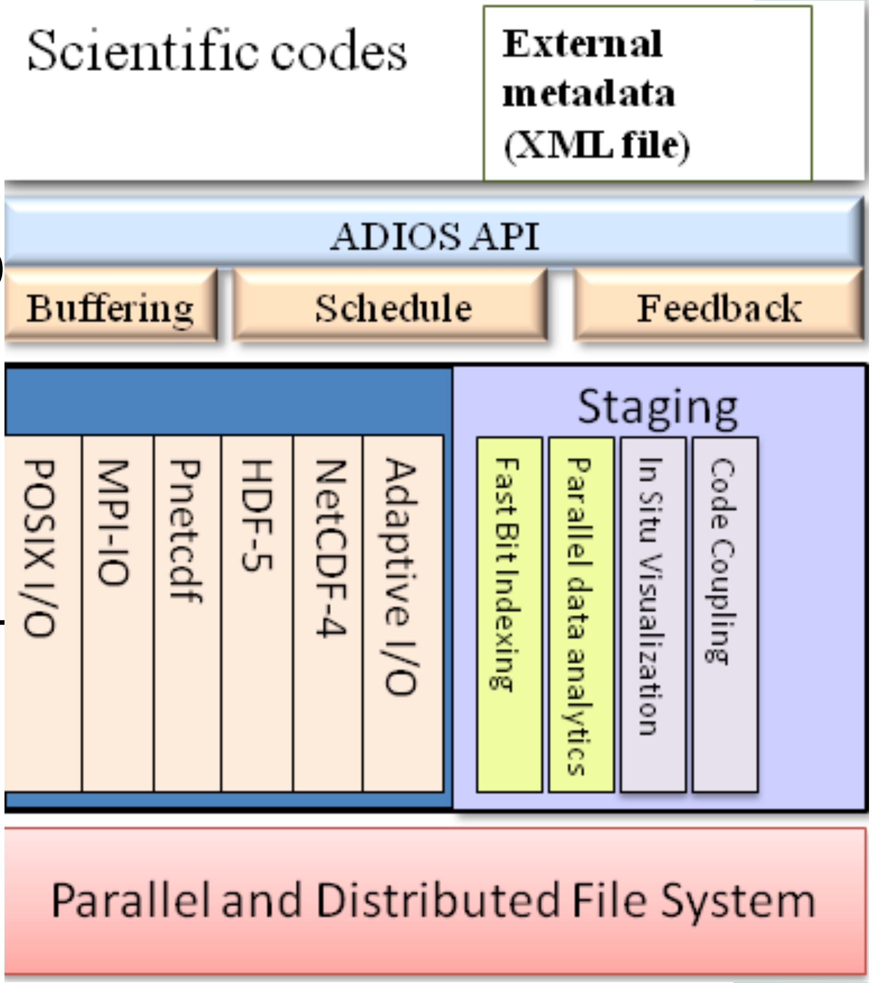
- End users should be able to select the most efficient I/O method for their code, with minimal effort in terms of code updates.
- Large-scale simulations should not be slowed down by I/O
- It is desirable to have I/O commands in codes independent of platform and file formats
  - Tools are needed to support asynchronous I/O (running concurrently with computations)
  - Default data formats should be flexible, efficient and robust
  - Tools are needed for allowing multiple I/O methods to be plugged-in through adaptable I/O libraries
- Performance-driven choices should not prevent data from being stored in the desired file format, since this is crucial for later data analysis.
- Have efficient ways of identifying and selecting certain data for analysis, to help end users cope with the flood of data being produced by these codes.
- Make it easy to introduce new research I/O methods, without changing your code.
- Make it easy to allow I/O to do more than just I/O → code coupling, in situ visualization.

Our approach.

- Componentize the I/O layer.
  - Similar to the approach PETSC took.
- Let I/O do “more than I/O”.
  - Synchronous output.
  - Asynchronous output.
  - Different file formats for output.
  - Code coupling.
  - In situ analysis.
  - In situ visualization.
- Design a new, metadata rich, file format for massively parallel file systems/computers.

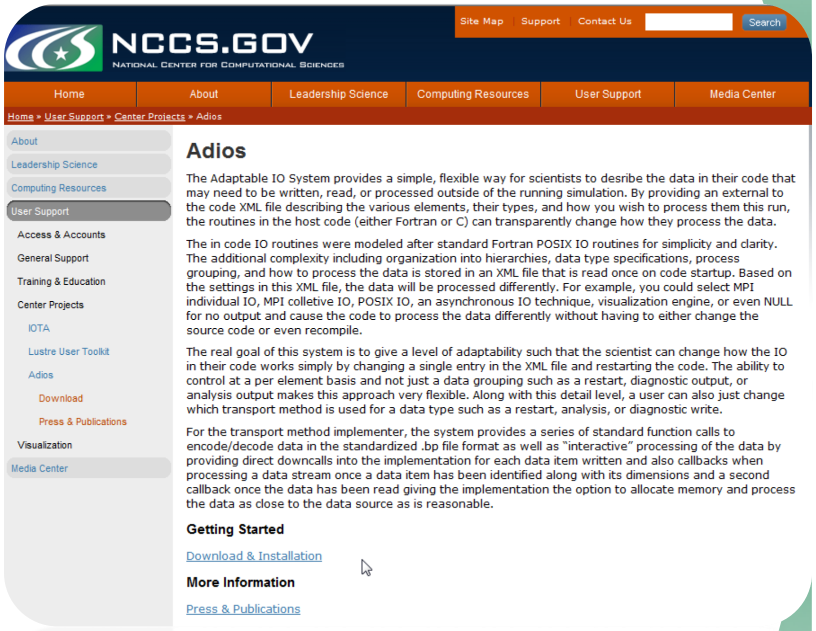
ADIOS

- Overview
  - Allows plug-ins for different I/O implementations
  - Abstracts API from the method used for I/O
- Simple API, almost as easy as F90 I/O
- Synchronous and asynchronous transports supported with no code changes
- Change I/O method by changing XML
  - Non XML version released in near future.
- ADIOS buffers data.
- ADIOS allows multiple transport methods per group



ADIOS 1.0: Open source

- http://www.nccs.gov/user-support/center-projects/adios/
- **ADIOS Ignites Combustion Simulations** http://www.hpcwire.com/features/ADIOS-Ignites-Combustion-Simulations-67321602.html
- **Fusion Gets Faster** http://www.hpcwire.com/features/Fusion-Gets-Faster-51820167.html?viewAll=y
- **Researchers Conduct Breakthrough Fusion Simulation** http://www.hpcwire.com/offthewire/Researchers_Conduct_Breakthrough_Fusion_Simulation.html
- Supports Mac (for reading; soon writing), Linux workstations, cluster, Cray XT, IBM BGP.



Adios

The Adaptable IO System provides a simple, flexible way for scientists to desribe the data in their code that may need to be written, read, or processed outside of the running simulation. By providing an external to the code XML file describing the various elements, their types, and how you wish to process them this run, the routines in the host code (either Fortran or C) can transparently change how they process the data.

The in code IO routines were modeled after standard Fortran POSIX IO routines for simplicity and clarity. The additional complexity including organization into hierarchies, data type specifications, process grouping, and how to process the data is stored in an XML file that is read once on code startup. Based on the settings in this XML file, the data will be processed differently. For example, you could select MPI individual IO, MPI colletive IO, POSIX IO, an asynchronous IO technique, visualization engine, or even NULL for no output and cause the code to process the data differently without having to either change the source code or even recompile.

The real goal of this system is to give a level of adaptability such that the scientist can change how the IO in their code works simply by changing a single entry in the XML file and restarting the code. The ability to control at a per element basis and not just a data grouping such as a restart, diagnostic output, or analysis output makes this approach very flexible. Along with this detail level, a user can also just change which transport method is used for a data type such as a restart, analysis, or diagnostic write.

For the transport method implementer, the system provides a series of standard function calls to encode/decode data in the standardized .bp file format as well as "interactive" processing of the data by providing direct downcalls into the implementation for each data item written and also callbacks when processing a data stream once a data item has been identified along with its dimensions and a second callback once the data has been read giving the implementation the option to allocate memory and process the data as close to the data source as is reasonable.

Getting Started

Download & Installation

More Information

Press & Publications

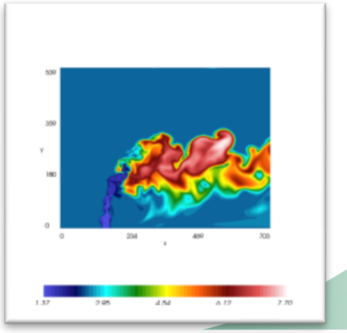
A few ADIOS Methods

- Posix
  - Writes 1 file per process in a ADIOS-BP file format.
- MPI
  - Writes 1 file in ADIOS-BP file format.
- MPI_STRIPE2
  - Lustre optimized, to set stripe size, count, transmission size with ADIOS-BP file format.
- PHDF5
  - Writes data in HDF5 file format.

BP File Format

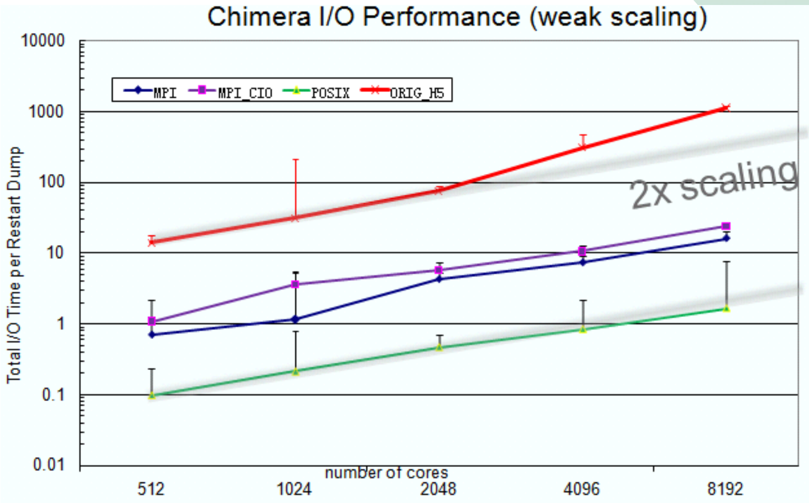
Process Group 1	Process Group 2	...	Process Group n	Process Group Index	Vars Index	Attributes Index	Index Offsets and Version #

- As in workflows, fault tolerance is critical for success of a parallel file format.
- Fully 64-bit, tested with files over 20TB, variables over 2 TB.
- Failure of single writer (even root) not fatal
- Necessary to have a hierarchical view of the data (like HDF5).
- Tested at scale (220K processors for XGC-1) with over 30TB in a single file.
- S3D code has generated over 100 TB in a few 'runs'.

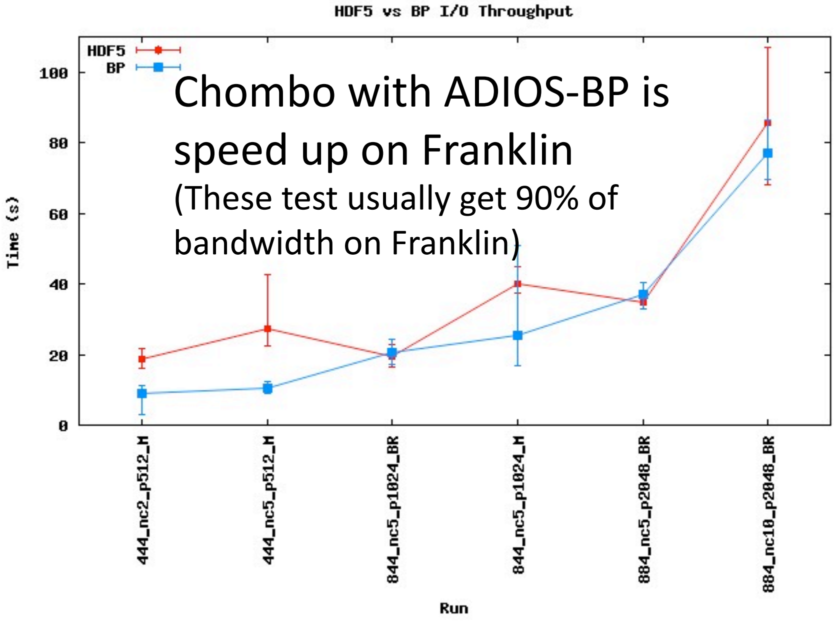


ADIOS Write Performance

- Introduce ADIOS.
- GTC: over 35 GB/s on Cray XT4
- XGC1: over 30 GB/s on XT4
- S3D: over 20 GB/s on XT4.
- Chimera 1000x better than apps first attempt.
    - But the apps people fixed this, no magic here… Also, Lustre fixed their problem



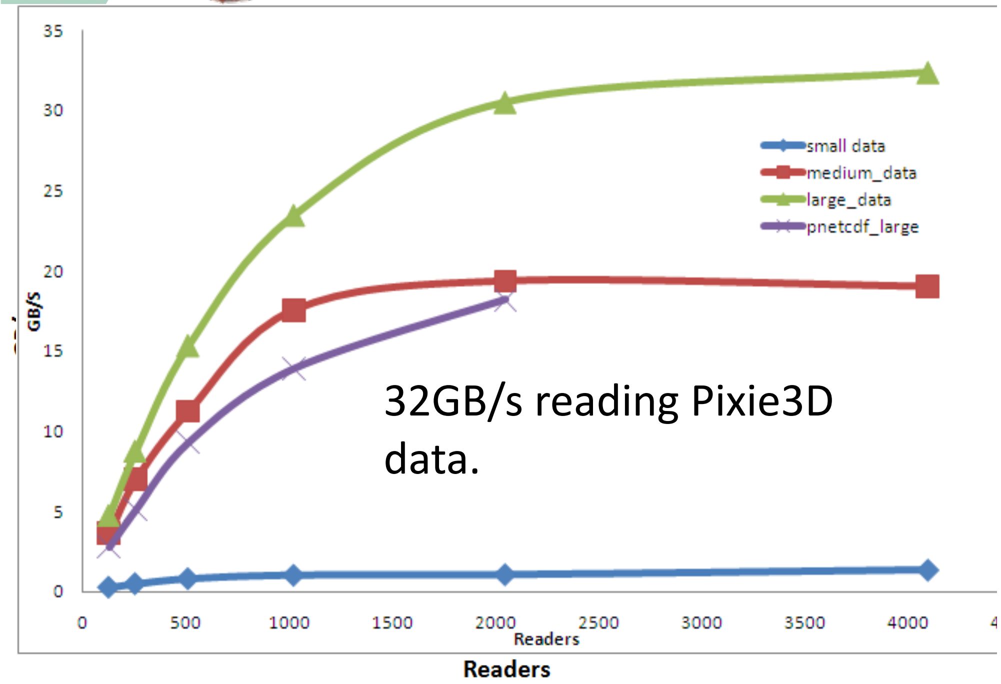
- Plot minimum value from 5 runs with 9 restarts/run
- Error bars show maximum time for the method.
- Parallel HDF5 was hand coded by Chimera team



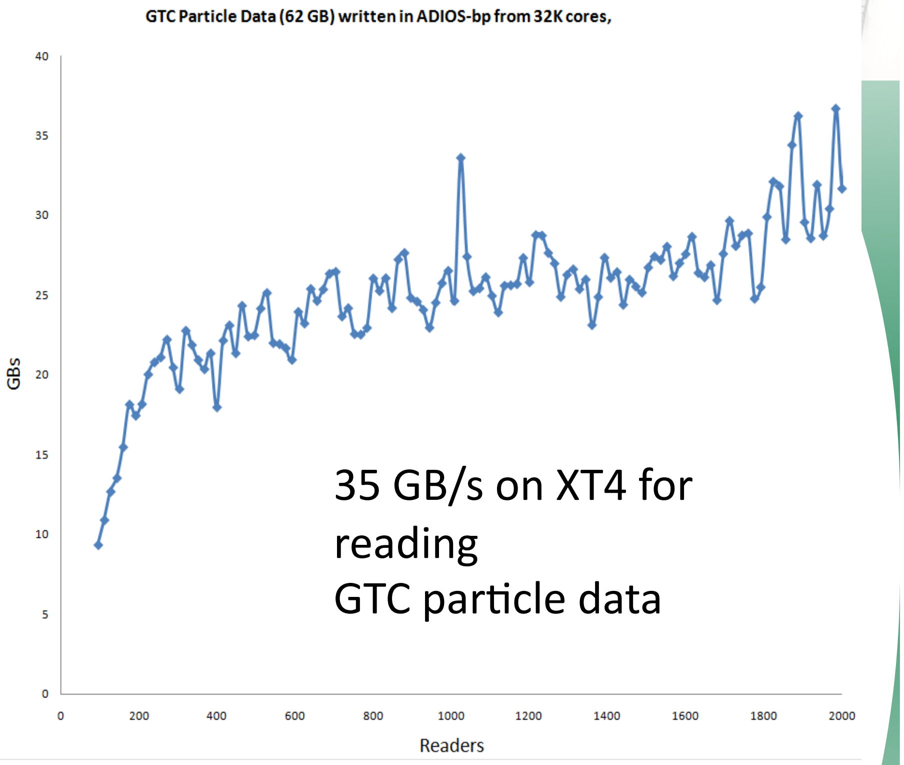
Chombo with ADIOS-BP is speed up on Franklin

(These test usually get 90% of bandwidth on Franklin)

ADIOS-BP Read performance on XT4



32GB/s reading Pixie3D data.



GTC Particle Data (62 GB) written in ADIOS-bp from 32K cores,

35 GB/s on XT4 for reading
GTC particle data

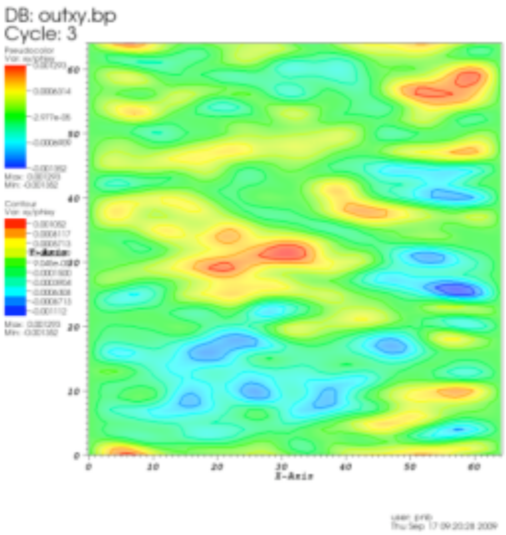
- We can read any subset (space/time) of data from any variable.
- APIs are easy to use.

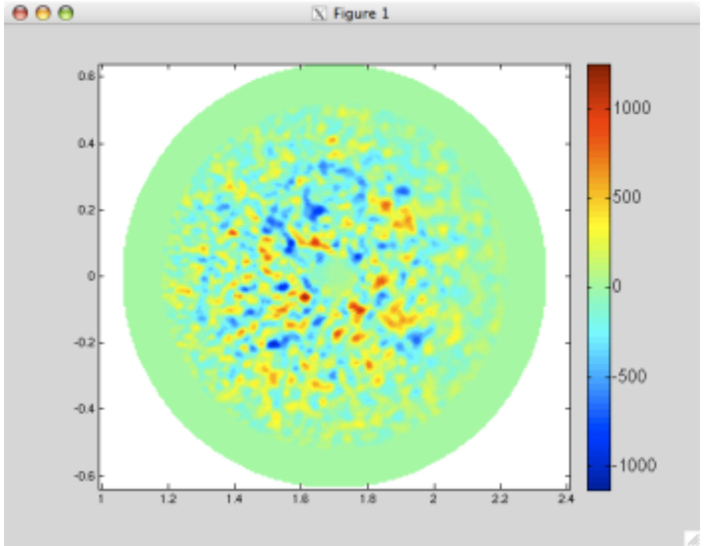
bpls (can extract any portion of data).

```
$ time /ccs/proj/e2e/pnorbert/ADIOS/ADIOS/trunk/utils/bpls/bpls -l record.bp -v
  of groups:     1
  of variables:  32
  of attributes: 0
  time steps:    10 starting from 1
  file size:     162 GB
  bp version:    1
Group record:
  double   /time           {10} = 0.003 / 0.03
  integer  /itime          {10} = 3 / 30
  integer  /nvar           scalar = 8
  integer  /dimensions/nxd+2  scalar = 1026
  integer  /dimensions/nyd+2  scalar = 514
  integer  /dimensions/nzd+2  scalar = 514
  double   /var/v1         {10, 514, 514, 1026} = 1 / 1
  double   /var/v2         {10, 514, 514, 1026} = -2.07946e-06 / 3.43263e-08
  double   /var/v3         {10, 514, 514, 1026} = -1.17581e-10 / 1.24015e-10
  double   /var/v4         {10, 514, 514, 1026} = -3.65092e-13 / 3.65092e-13
  double   /var/v5         {10, 514, 514, 1026} = -7.95953e-11 / 7.95953e-11
  double   /var/v6         {10, 514, 514, 1026} = -0.184178 / 0.0123478
  double   /var/v7         {10, 514, 514, 1026} = -0.000488281 / 0.984914
  double   /var/v8         {10, 514, 514, 1026} = 0 / 0
  byte     /name/v1_name     {20} = 32 / 111
  byte     /name/v2_name     {20} = 32 / 94
  byte     /name/v3_name     {20} = 32 / 94
  byte     /name/v4_name     {20} = 32 / 94
  byte     /name/v5_name     {20} = 32 / 94
  byte     /name/v6_name     {20} = 32 / 94
  byte     /name/v7_name     {20} = 32 / 94
  byte     /name/v8_name     {20} = 32 / 101
  integer  /bconds          {48} = -4 / 7
real    0m2.091s
```

ADIOS characteristics are constantly being added. Criteria is that they (sum of all characteristics) take <0.1% of the I/O time.

ADIOS BP Visit & Matlab Readers





- **rz=adiosread(meshfile,'/coordinates/values');**
- **var=adiosread(pfile,'pot3d','/node_data[1]/values');**
- Visit BP reader is parallel. → Pugmire/Ahern.

ADIOS 1.2

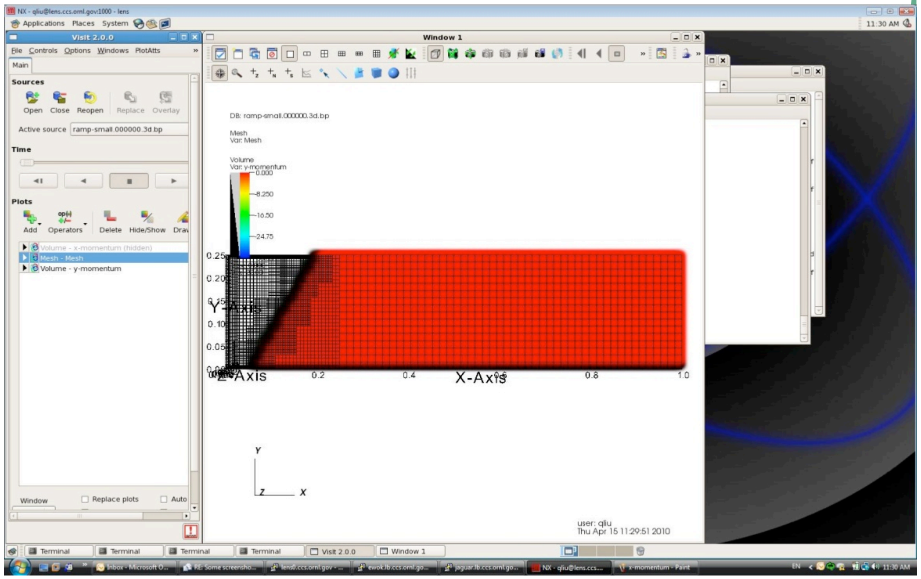
July 11, 2010

New characteristics into ADIOS-BP

- Histograms can be automatically generated, in the footer (no added cost in writing).
  - <analysis group="temperature" var="temperature" break-points="0, 100, 200, 300" />
  - <analysis group="temperature" var="temperature" min="0" max="300" count="3"/>
  - Both the above inputs create bins [0, 100), [100, 200), [200, 300)
- Min/max over time steps.
- Averages.
- Easy to add new characteristics.

ADIOS (NO XML)

- ADIOS 1.2 contains APIs for users who don't wish to use the XML
  - adios_init_local
  - adios_allocate_buffer
  - To declare a ADIOS group
  - adios_declare_group
  - To select a I/O method for a ADIOS group
  - adios_select_method
  - To define a ADIOS variable
  - adios_define_var



Active source ramp-small.000000.3d.bp

Volume - x-momentum (hidden)
Mesh - Mesh
Volume - y-momentum

DB: ramp-small.000000.3d.bp

Mesh
Var: Mesh

Volume
Var: y-momentum

user: qliu
Thu Apr 15 11:29:51 2010

New ADIOS 1.2 methods

- NC4
  - Built on top on hdf5 with parallel hdf5 extensions.
  - Maintains NC3 compatibility.
  - Don’t take advantage of groups, etc.
- NSSI
  - Data staging using the Sandia method.
- DataTap
  - Data staging from Georgia Tech.
- Data Spaces
  - Uses the DART transport for code coupling.
- AMR
  - Optimized for AMR codes, and codes with small writes.
- Adaptive
- BGP
  - Optimized for BGP synchronous writes.

ADIOS_AMR Method

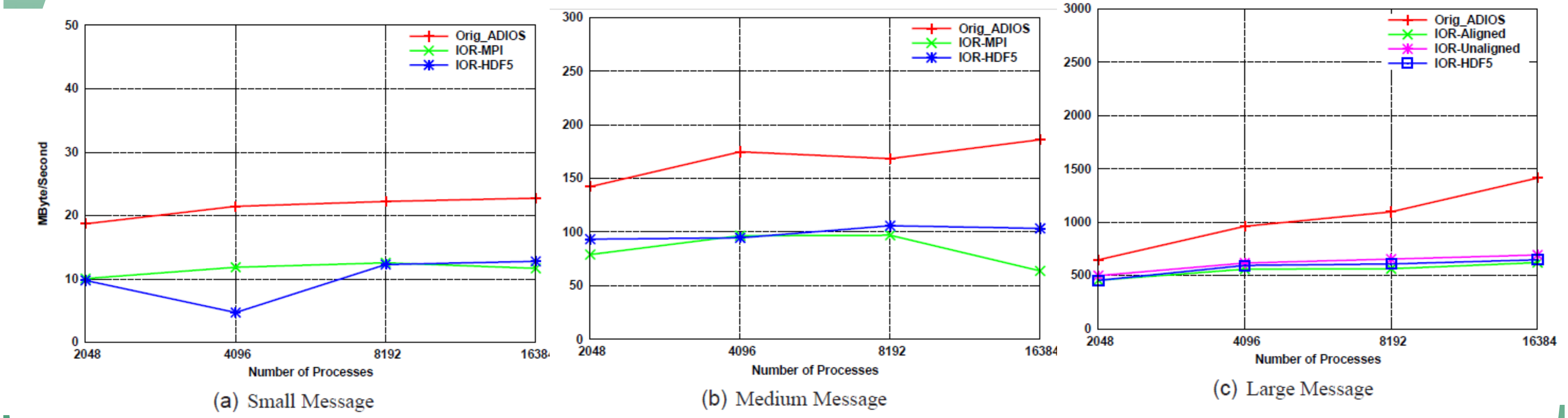
- Targeted specifically at AMR codes, and codes with lots of small writes.
- In AMR code, each processor can output varied amount of (possibly small) data.
  - Dynamic aggregation technique used to achieve good I/O performance.
- Initial results on Cray XT5, 57600 processors with 8MB/proc on average, striped on 600 OST's.
- Recent tests for the S3D code
  - 120,000 cores, 5 arrays (several scalars), 28^3 variables (doubles)/write. (800 KB/MPI proc)
  - Results are 5.5s +- 1s, ~16GB/s = 0.3% overhead
  - Run with 60^3 elements = 17s +- 2s, ~53 GB/s = 1% overhead.
  - **ALL TIMES include open, write, close, flush.**

Adaptive Method

- New adaptive method meant to handle the variability of the writes.

But what about IBM BGP (Intrepid)

- No Changes in ADIOS…
- Write data from a 3D domain decomposition.
- Small = 128 KB, Medium = 1MB, Large = 8 MB (per mpi process)



(a) Small Message

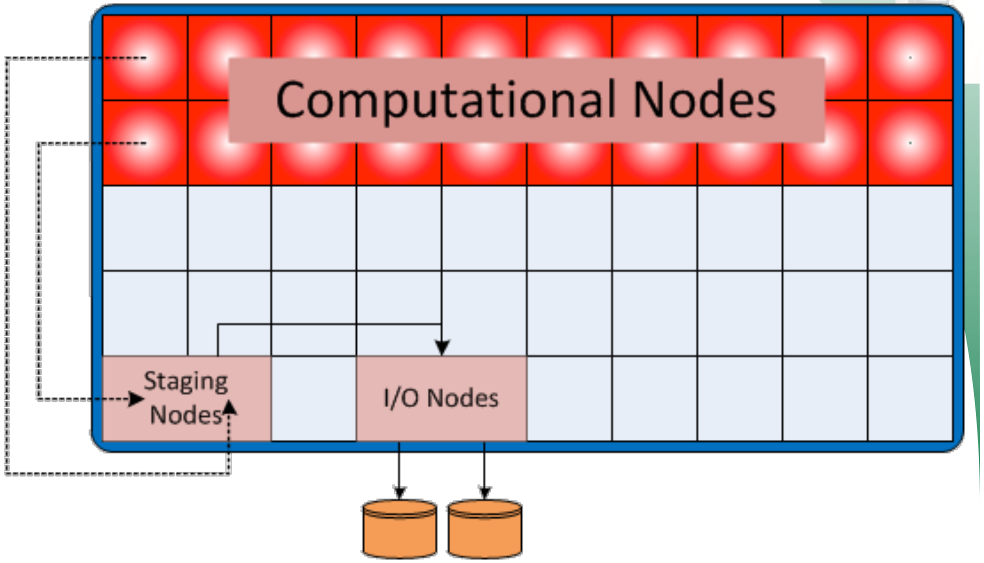
(b) Medium Message

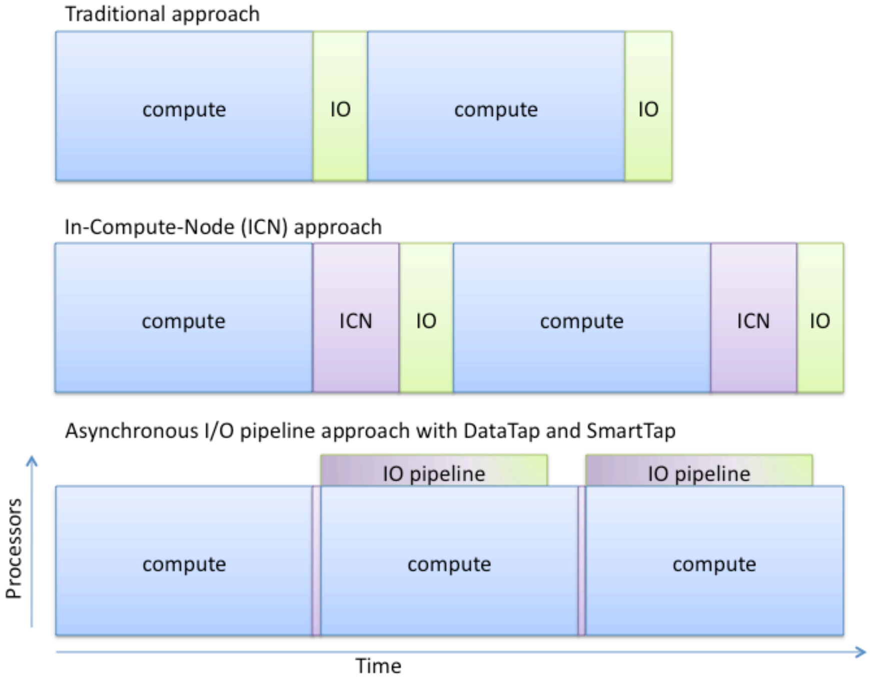
(c) Large Message

PVFS

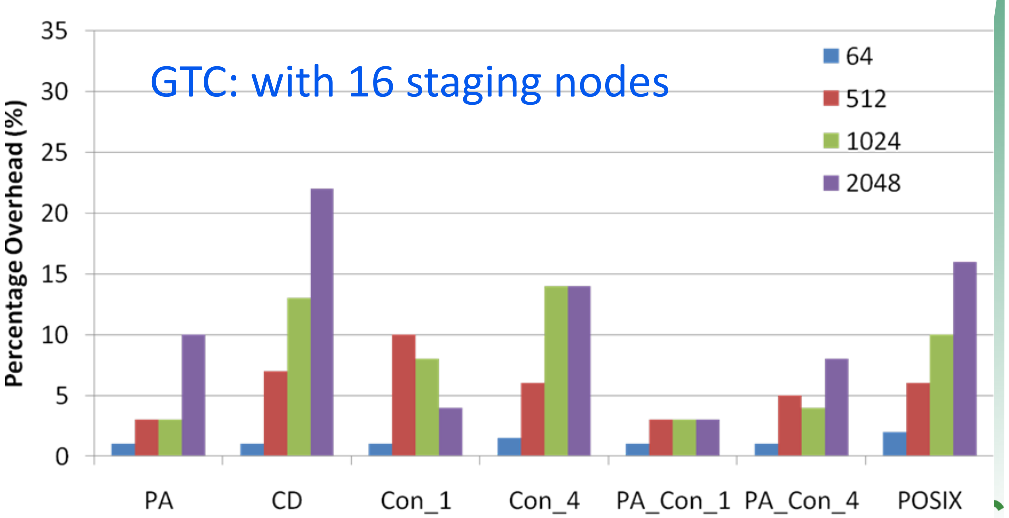
Staging methods

- DataTap (Georgia Tech)
- NSSI (Sandia)





Scheduling of I/O is essential

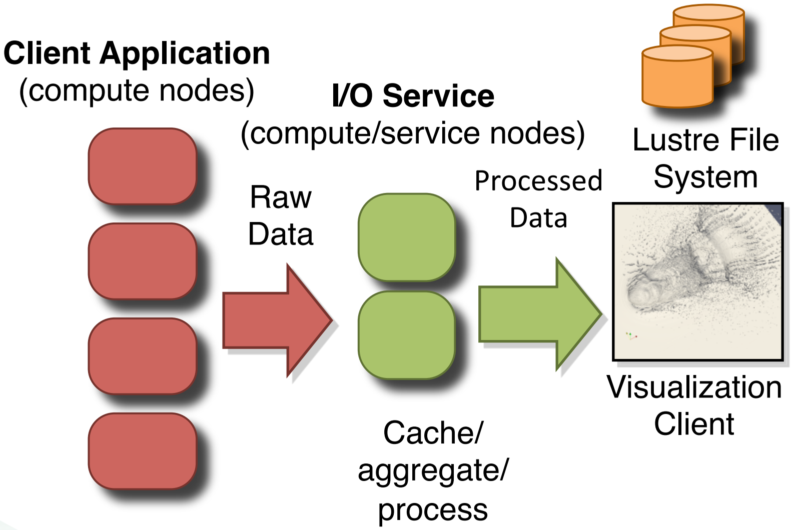


NSSI staging method (Sandia: Oldfield, Kordenbrock

- Staging method + server built for caching + aggregation on sever side.
- NESSI transport
  - RPC layer on portals and infiniband.
  - Mostly tested with NC4 method to do the
  - Will support any adios method to write.

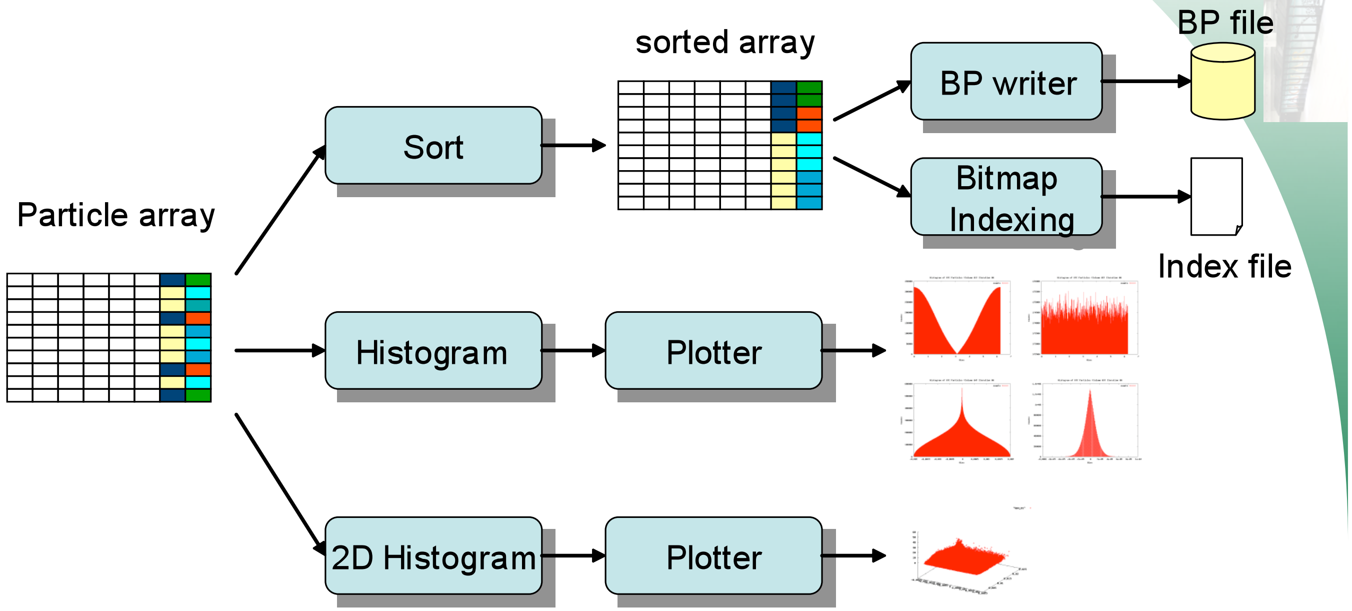
Nessie

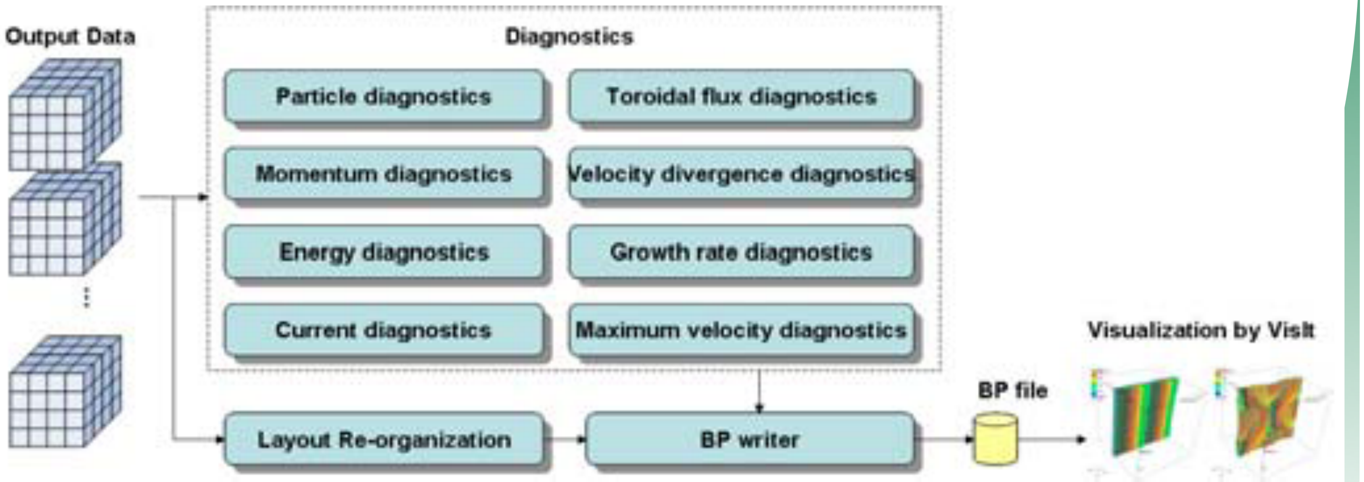
NEtwork-Scalable Service InterfacE



I/O for more than just I/O

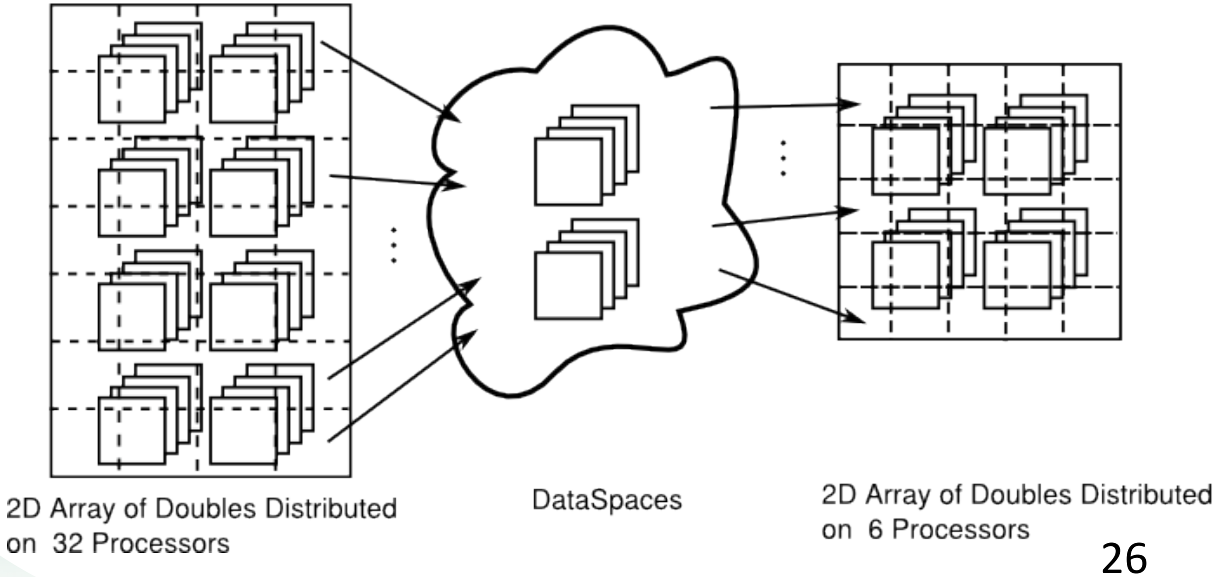
- Use the staging nodes and create a workflow in the staging nodes.
- Allows us to explore many research aspects.
- Improve total simulation time by 2.7% over synchronous writes to disk.





XGC-0 – M3D-MPP Coupling Using DataSpaces

- The simulations exchange multi-dimensional data arrays (e.g., 2D)
  - Domain discretization is different for the two applications
  - Data redistribution is transparent and implicit through the space
- The simulations have different interaction patterns
  - e.g., one-to-many, many-to-many, many-to-one


2D Array of Doubles Distributed on 32 Processors
DataSpaces
2D Array of Doubles Distributed on 6 Processors

How does it work?

Code which sends data

```
call adios_open (adios_handle,
 "writer2D", fn, "w", group_comm,
 adios_err)
#include "gwrite_writer2D.fh"
call adios_close (adios_handle,
 adios_err)
```

- Generate the XML file to map F90/C variables to names.

```
<adios-group name="writer2D" >
 <global-bounds
 dimensions="dim_x_global,dim_y_global“
 offsets="offs_x,offs_y">
 <var name="xy" type=“real“
 dimensions="dim_x_local,dim_y_local"/>
</global-bounds>
</adios-group>
<transport group="writer2D" method = “DART” ’>
```

Code which receives data

```
call adios_set_read_method (: DART  ,ierr)
call adios_read_init (group_comm, ierr)
call adios_fopen (fh, fn, group_comm, gcnt,
    adios_err)
call adios_gopen (fh, gh, "writer2D", vcnt,
    acnt, adios_err)
call adios_read_var (gh, "dim_x_global",
    offset, readsize, dim_x_local,
    read_bytes)
call adios_read_var (gh, "dim_y_global",
    offset, readsize, dim_y_local,
    read_bytes)
call adios_read_var (gh, "xy", offset,
    readsize, xy, read_bytes)
call adios_gclose (gh, adios_err)
call adios_fclose (fh, adios_err)
```

Now we have memory to memory coupling

Everything can happen with APIs too

ADIOS 1.1

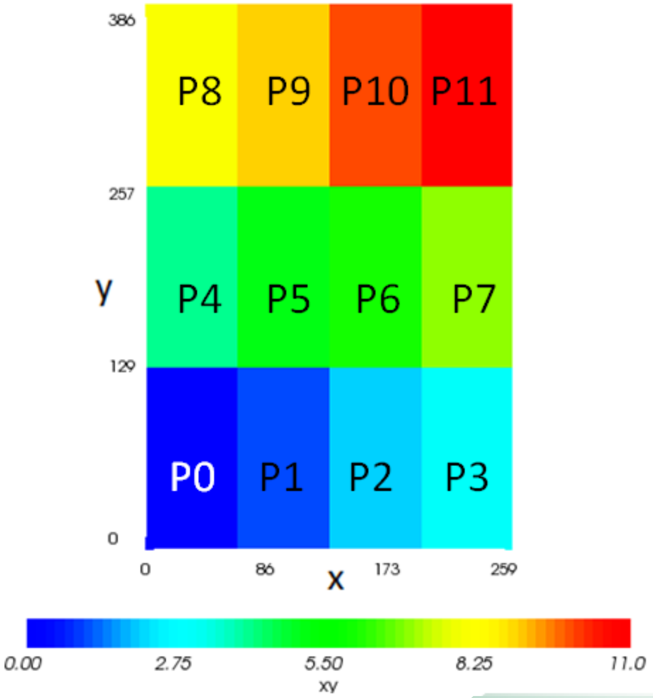
DEMO

Loading ADIOS on jaguar/ewok/lens

- Step 1, load adios
  - module load adios/1.1.0
- Can build adios from the source
  - http://www.nccs.gov/user-support/center-projects/adios/download/
  - Must have MPI installed, and MinixML installed.
  - http://www.minixml.org/software.php

Write Example

- In this example you will start with a 2D code which writes data with a 2D array, with a 2D domain decomposition, as shown in the figure.
  - `xy = 1.0*rank + 1.0*ts`
- We will write out 2 time-steps, in separate files.
- For simplicity, we will work on only 12 cores, arranged in a 4 x 3 arrangement.
- Each processor will allocate a 65x129 array (xy).
- The total size of the array = 4*65, 3*129



Looking at I/O portion of coupling_writer_2D_base.F90

```
posx = mod(rank, npx) ! 1st dim easy: 0, npx, 2npx... are in the same X position
posy = rank/npx ! 2nd dim: npx processes belong into one dim
offs_x = posx * ndx ! The processor offset in the x dimension for the global dimensions
offs_y = posy * ndy ! The processor offset in the x dimension for the global dimensions
nx_local = ndx ! The size of data that the processor will write in the x dimension
ny_local = ndy ! The size of data that the processor will write in the y dimension
nx_global= npx * ndx ! The size of data in the x dimension for the global dimensions
ny_global= npy * ndy ! The size of data in the y dimension for the global dimensions

do ts=0,timesteps-1
   write(filename,'(a4,i2.2,a4,i2.2)') 'cpes',ts,'.bn.',rank ! The name of each output file 1/proc
   xy = 1.0*rank + 1.0*ts ! The value to place in the xy array
   open(100,file=filename,status='UNKNOWN',form='unformatted',action='write')
   write(100) nx_global,ny_global
   write(100) nx_local,ny_local
   write(100) xy
   close(100)
end do
```

Compiling and running the code

- Run the code, see 24 files produced from 12 processors

```
Make
> make
> mpirun -np12 ./coupling_writer_2D_base
 ts= 0
 ts= 1
> ls *.bn*
cpes00.bn.00 cpes00.bn.05 cpes00.bn.10 cpes01.bn.03
   cpes01.bn.08
cpes00.bn.01 cpes00.bn.06 cpes00.bn.11 cpes01.bn.04
   cpes01.bn.09
cpes00.bn.02 cpes00.bn.07 cpes01.bn.00 cpes01.bn.05
   cpes01.bn.10
cpes00.bn.03 cpes00.bn.08 cpes01.bn.01 cpes01.bn.06
   cpes01.bn.11
cpes00.bn.04 cpes00.bn.09 cpes01.bn.02 cpes01.bn.07
```

ADIOS the code -1

1. cp coupling_writer_2D_base.F90 coupling_writer_2D.F90, edit coupling_writer_2D.F90
2. Uncomment lines 22-24. ! character (len=200) :: group
   - We need to declare variables to use for ADIOS.
   - Since ADIOS is 64-bit, the variables are integer*8
3. Line 32: We need to initialize ADIOS: like MPI_Init, after call MPI_Comm_size
   - call adios_init ('coupling2D_writer.xml', ierr)
4. Lines 41,68: Need to finalize ADIOS: before MPI_Finalize
   - call adios_finalize (rank, adios_err)
5. Line 57: replace the output file name.
   - write(filename,'(a4,i2.2,a3)') 'cpes',ts,'.bp'
6. Line 60: replace F90 open with ADIOS open
   - call adios_open (adios_handle, 'writer2D', trim(filename), 'w', group_comm, adios_err)
7. Line 64: replace the close with the adios_close
   - call adios_close (adios_handle, adios_err)
8. Line 61-63: replace the writes with the ADIOS include, # starts at first column in file.
   - #include "gwrite_writer2D.fh"

OAK RIDGE National Laboratory

ADIOS

The ADIOS XML configuration file.

- Describe each IO grouping.
- Maps a variable in the code, to a variable in a file.
- Map an IO grouping to transport method(s).
- Define buffering allowance
- "XML-free" API completed and included in ADIOS 1.2

XML Overview

- Look at the original I/O
  - write(100) nx_global,ny_global
  - write(100) nx_local,ny_local
  - write(100) xy
- Look at coupling2D_writer.xml
- <adios-group name=“writer2D”>
- <var name=“nx_global” type=“integer”/>
- <var name=“ny_global” type=“integer”/>

- <var name=“nx_local” path=“/aux” type=“integer”/>
- <var name=“ny_local” path=“/aux” type=“integer”/>
- <var name=“xy” type=“real*8” dimensions=“nx_local,ny_local”/>
- </adios-group>

XML overview (global array)

- We want to read in xy from an arbitrary number of processors, so we need to write this as a global array.
- Need 2 more variables, to define the offset in the global domain
  - <var name=“offs_x” path=“/aux” type=“integer”/>
  - <var name=“offs_y” path=“/aux” type=“integer”/>
- Need to define the xy variable as a global array
  - Place this around the lines defining xy in the XML file.
  - <global-bounds dimensions="nx_global,ny_global" offsets="offs_x,offs_y">
  - </global-bounds>

XML overview

- Need to define the method, we will use MPI.
  - <transport group="writer2D" method="MPI"/>
- Need to define the buffer
  - <buffer size-MB="4" allocate-time="now"/>
  - Can use any size, but if the buffer > amount to write, the I/O to disk will be faster.
- Need to define the host language (C or Fortran ordering of arrays).
  - <adios-config host-language="Fortran">
- Set the XML version
  - <?xml version="1.0"?>
- And end the configuration file
  - </adios-config>

The final XML file

```
<?xml version="1.0"?>
<adios-config host-language="Fortran">

 <adios-group name=“writer2D”>
 <var name=“nx_global” type=“integer”/>
 <var name=“ny_global” type=“integer”/>

 <var name=“offs_x” path=“/aux” type=“integer”/>
 <var name=“offs_y” path=“/aux” type=“integer”/>
 <var name=“nx_local” path=“/aux” type=“integer”/>
 <var name=“ny_local” path=“/aux” type=“integer”/>

 <global-bounds dimensions="nx_global,ny_global" offsets="offs_x,offs_y">
 <var name=“xy” type=“real*8” dimensions=“nx_local,ny_local”/>
 </global-bounds>

 </adios-group>

<transportgroup="writer2D" method="MPI"/>
<buffer size-MB="4" allocate-time="now"/>

</adios-config>
```

gpp.py

- Converts the XML file into F90 (or C) code.
- > gpp.py coupling2D_writer.xml
- > cat gwrite_writer2D.fh

```
adios_groupsize = 4 &
 + 4 &
 + 4 &
 + 4 &
 + 4 &
 + 4 &
 + 8 * (nx_local) * (ny_local)
call adios_group_size (adios_handle, adios_groupsize, adios_totalsize, adios_err)
call adios_write (adios_handle, "nx_global", nx_global, adios_err)
call adios_write (adios_handle, "ny_global", ny_global, adios_err)
call adios_write (adios_handle, "offs_x", offs_x, adios_err)
call adios_write (adios_handle, "offs_y", offs_y, adios_err)
call adios_write (adios_handle, "nx_local", nx_local, adios_err)
call adios_write (adios_handle, "ny_local", ny_local, adios_err)
call adios_write (adios_handle, "xy", xy, adios_err)
```

Compile and run the code

```
> make
>  mpirun -np 12 ./coupling_writer_2D
 ts= 0
 ts= 1
>  ls *.bp
cpes00.bp cpes01.bp
```

- Now we change the transport method to POSIX
- Now we change the transport method to phdf5

ADIOS Tools

- bpls
  - Similar to h5dump/ncdump
  - Also shows array min/max values
  - Performance independent of data size
- bp2h5, bp2ncd
  - Convert BP format into HDF5 or NetCDF

ADIOS Reading

- GOALS
  - Learn how to look at a ADIOS-BP file.
  - Learn how to convert a code to read in ADIOS files.
  - Learn how to read in data from an arbitrary number of processors.

bpls

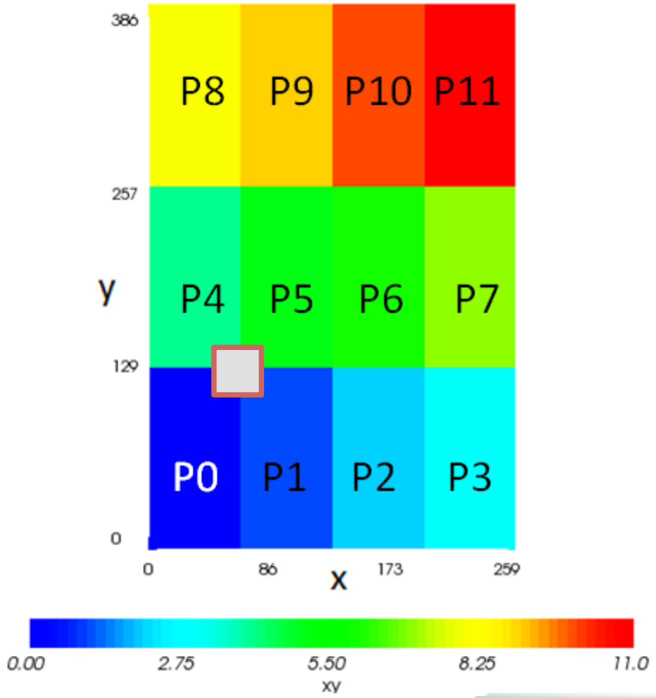
```
> bpls -lv cpes00.bp
File info:
 of groups: 1
 of variables: 7
 of attributes: 0
 time steps: 1 starting from 1
 file size: 795 KB
 bp version: 1
 endianness: Little Endian

Group writer2D:
 integer /nx_global scalar = 260
 integer /ny_global scalar = 387
 integer /aux/offs_x scalar = 0
 integer /aux/offs_y scalar = 0
 integer /aux/nx_local scalar = 65
 integer /aux/ny_local scalar = 129
 double /xy {387, 260} = 0 / 11
```

bpls

- Use bpls to read in a 2D slice
-
```
> bpls cpes00.bp -d xy –n 2 -s “128,64” -c “2,2”
```

```
 double /xy {387, 260}
 slice (128:129, 64:65)
(128,64)    0 1
 (129,64)    4 5
```



bp2h5, bp2ncd

```
• > module load hdf5
• > module load netcdf
• > bp2h5 cpes00.bp
• > h5ls cpes00.h5
    aux Group
    nx_global Dataset {SCALAR}
    ny_global Dataset {SCALAR}
    xy Dataset {387, 260}
• > bp2ncd cpes00.bp
• > ncdump –h  cpes00.nc
    netcdf cpes00 {
    dimensions:
         nx_global = 260 ;
         ny_global = 387 ;
         aux_nx_local = 65 ;
         aux_ny_local = 129 ;
         aux_offs_x = 65 ;
         aux_offs_y = 129 ;
    variables:
         double xy(nx_global, ny_global) ;
    }
```

Looking at I/O portion of coupling_reader_2D_base.F90

- We loop over the 2 timesteps to read, and write out the results 1 ascii file/reader

```
do ts = 0, ntsteps-1                                                   ! Loop for ts=0,1
 write(filename,'(a4,i2.2,a4,i2.2)') 'cpes',ts,'.bn.',rank             ! Get the filename
 open(100,file=filename,status='OLD',form='unformatted',action='read')  !open the file
 read(100) nx_global,ny_global                                          !start reading
 read(100) readsize(1), readsize(2)                                     ! Size of array xy to read
 offset(1) = mod(rank, posx) * readsize(1)                              !calculate offsets for
 offset(2) = rank/posx * readsize(2)                                    !writing in ascii file
 allocate( xy (readsize(1), readsize(2)) )                              !allocate the memory

 read(100) xy                                                           !read in the big array
                                                       !dump out the array in 12 separate files
 do j=1,readsize(2)
 do i=1,readsize(1)
 write (200+rank, '(3i5,f8.1)'), ts,i-1+offset(1),j-1+offset(2),xy(i,j)
 enddo
 enddo
 close(100)                                                             !close the file
```

Compile and run the code

```
> make
> mpirun -np 12 ./coupling_reader_2D_base
> ls fort.*
```

```
fort.200 fort.202 fort.204 fort.206 fort.208 fort.210
fort.201 fort.203 fort.205 fort.207 fort.209 fort.211
```

```
> tail fort.211
```

```
1 250 386 12.0
1 251 386 12.0
1 252 386 12.0
1 253 386 12.0
1 254 386 12.0
1 255 386 12.0
1 256 386 12.0
1 257 386 12.0
1 258 386 12.0
1 259 386 12.0
```

- File writes timestep, x (global), y(global), xy

How to place ADIOS APIs into the read code

1. cp coupling_reader_2D_base.F90 coupling_reader_2D.F90
2. Line 13, Make offset and readsize integer*8, since ADIOS=64 bit.
3. Uncomment Lines 26-27 (ADIOS integer, ADIOS integer*8)
   - Need to declare variables that can tell us the number of groups, variables, attributes in a file. We also need file and group pointers.
4. Comment Line 29: We don't need this anymore. (integer :: posx=4)
5. Line 39: change the filename, since we have 1 file
   - write(filename,'(a4,i2.2,a3)') 'cpes',ts,'.bp'
6. Lines 40-41, replace the open statement with
   - call adios_fopen (fh, filename, group_comm, gcnt, ierr)
   - call adios_gopen (fh, gh, 'writer2D', vcnt, acnt, ierr)
7. Uncomment Line 43 ! If (ts==0) then
8. Replace lines 44-47 with adios_read_var calls, from read(100) nx_global
   - call adios_read_var(gh, 'nx_global', offset, readsize, nx_global, read_bytes)
   - call adios_read_var(gh, 'ny_global', offset, readsize, ny_global, read_bytes)
   - readsize(1) = nx_global / nproc !don't need to read in readsize(1)
   - readsize(2) = ny_global !don't need to read in readsize(2)

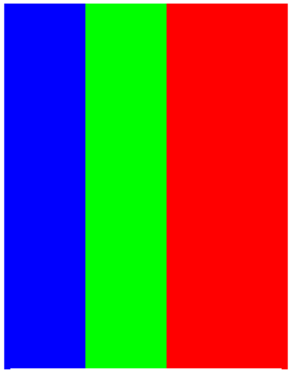
ADIOS the code

9. Lines 49-50, (we will read the data with a 1D domain decomposition). (changing the base offsets)
   - offset(1) = rank * readsize(1)
   - offset(2) = 0
10. Uncomment Lines 52-54 (since last proc might need to read in more data). (if (rank ==nproc-1) then …
11. Comment Line 55, since we have this from change 6. if (ts==0)
12. Line 59, change the read(100) xy to
    - call adios_read_var(gh, 'xy', offset, readsize, xy, read_bytes)
17. Line 66-67, replace the close(100) statement with
    - call adios_gclose (gh, ierr)
    - call adios_fclose (fh, ierr)

Compile and run the code

```
> make
> mpirun -np 1 ./coupling_reader_2D
> ls fort.*;tail –n 4 fort.100
```

```
fort.100
 1 256 386 12.0
 1 257 386 12.0
 1 258 386 12.0
 1 259 386 12.0
```



```
> mpirun -np 7 ./coupling_reader_2D
> ls fort.*;tail –n 4 fort.100
```

```
fort.100 fort.101 fort.102 fort.103 fort.104 fort.105 fort.106
 1 33 386 9.0
 1 34 386 9.0
 1 35 386 9.0
 1 36 386 9.0
```

- We can read in data from 1 – 260 processors now with a 1D domain decomp.

Conclusions.

- ADIOS is an I/O componentization framework that
  - Has been proven for extreme scale performance on massively parallel systems (real codes with >200K cores simulations).
  - Allows for both synchronous and asynchronous I/O transports.
  - Contains a new, metadata rich, I/O format that can allow for extreme scale I/O (largest runs at 220K cores for the XGC1 code now) and data analysis in situ with the computation.
- The ADIOS BP file format is a log-file format, which has shown extreme scalability for both write and read access.
- ADIOS 1.1 is available at http://www.nccs.gov/user-support/center-projects/adios/
- ADIOS 1.2 will be coming July 11, 2010

OAK RIDGE National Laboratory

ADIOS